UNBC CPSC 101 Team Term Project Winter 2018

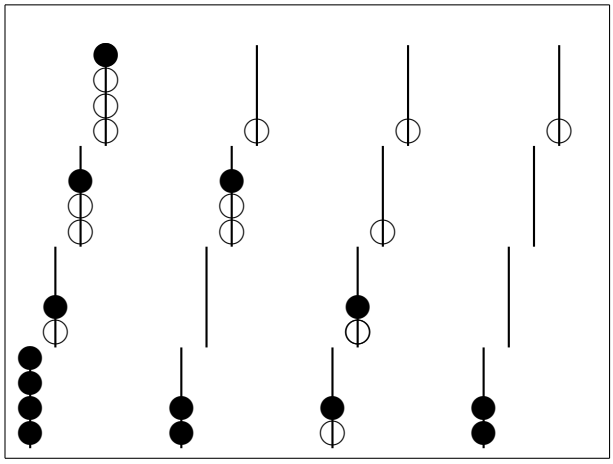

Figure 1: Lines in a Score 4 game.

The board shown to the left has four black lines and two white ones. The position shown could never arise in a real game.

Note

This problem specification is still imcomplete. Further details will be forthcoming.

Problem statement

The goal of this project is to write programs to play a 3-dimensional tic-tac-toe game that is sold commercially as "Score 4".[1]

The Game The game is played on a board which consists of a 4×4 grid of pegs (thin metal spikes). On each spike you can slide as many as four coloured beads. There are two players: White and Black. White has 32 white beads, and Black has 32 black beads. The players alternate taking turns, with White playing first. On each player's turn, the player places a bead on one of the 16 pegs that has less than 4 beads already on it.

The first player to get 4 beads in a straight line wins. Should the entire board be filled before either player completes a line, the game is a draw. (In fact, once it becomes clear that it is mathematically impossible for either player to win, the referee can declare the game to be a draw.) A straight line can be horizontal, vertical, or diagonal. Diagonals can be singly or doubly "skewed" as shown in Figure . If you count carefully, you should find that there are 76 lines on a completely-filled board.

Beads cannot hang in the air; they slide down to the bottom of the peg. This makes Score 4 different from $4 \times 4 \times 4$ tic-tac-toe.

Programming tasks

Your team needs to write two or three separate packages: a "referee", a human player interface, and a computer opponent. You may choose to combine the referee package

[1] see `http://www.boardgamegeek.com/game/3656`. It appears that this game was first produced in the 1970's, so patents have likely expired.

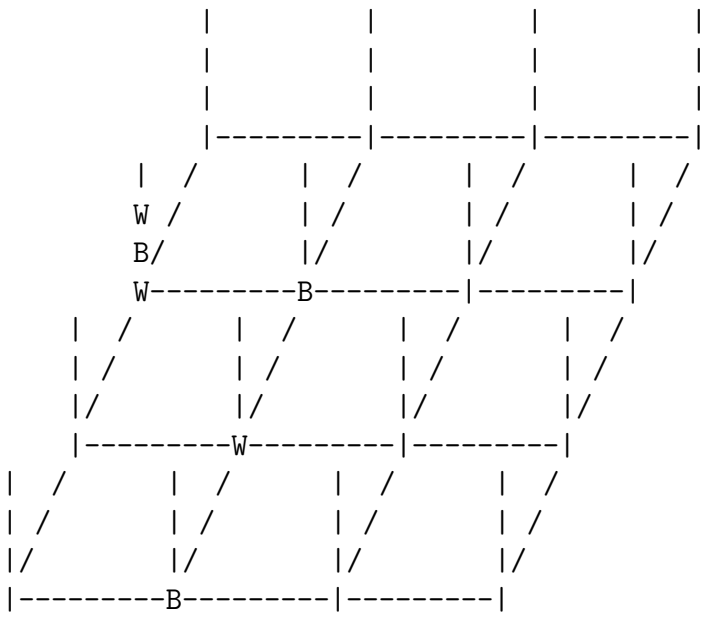

Figure 2: ASCII rendition of Score 4 board.

and human player interface package if you so choose. The details of how packaging work are subject to change and finalization.

The Referee The referee program allows a person to play Score 4 against the computer. It keeps track of whether either player has won the current game; lets the human player quit or restart the game any time (s)he chooses; and draws the board so that the human player can see the current game situation.

ASCII-graphics such as those shown in Figure 2 are acceptable for displaying the board, and were used in the past when this project was done in C++. However, a Graphical User Interface is preferred.

The exact mechanism that the referee program uses for interacting with the computer opponent are yet to be finalized and will be explained in more detail in a later handout.

The Computer Opponent The computer opponent is a stand-alone package that satisfies a common interface for all of the team's computer opponents. This allows the computer opponents to be captured in separate `.jar` files, and for a tournament between the various teams' computer opponents at the end of the semester.

The common interface satisfied by the computer opponents is yet to be specified.

General comments

The referee program and the computer opponent program make use of similar concepts, so they should make use of common classes. *Your coding will be graded in part on how much code is shared between the two packages;* as one of the goals of good object oriented programming is to create classes and objects that can be reused.

For the computer opponent program, correctness is far more important than cleverness. The computer opponent program must work correctly, even when used by a referee program other than your own. However, intelligent play by the computer opponent is not necessary, and should not be a priority when completing the project.

Dr David Casperson
2018-01-12